Title:	Portable Data Parallel Visualiza0on Algorithms with VTK-m
Author(s):	Lu, Kewei Canada, Curtis Vincent
Intended for:	Web
Issued:	2015-10-05

Portable Data Parallel Visualization Algorithms with VTK-m

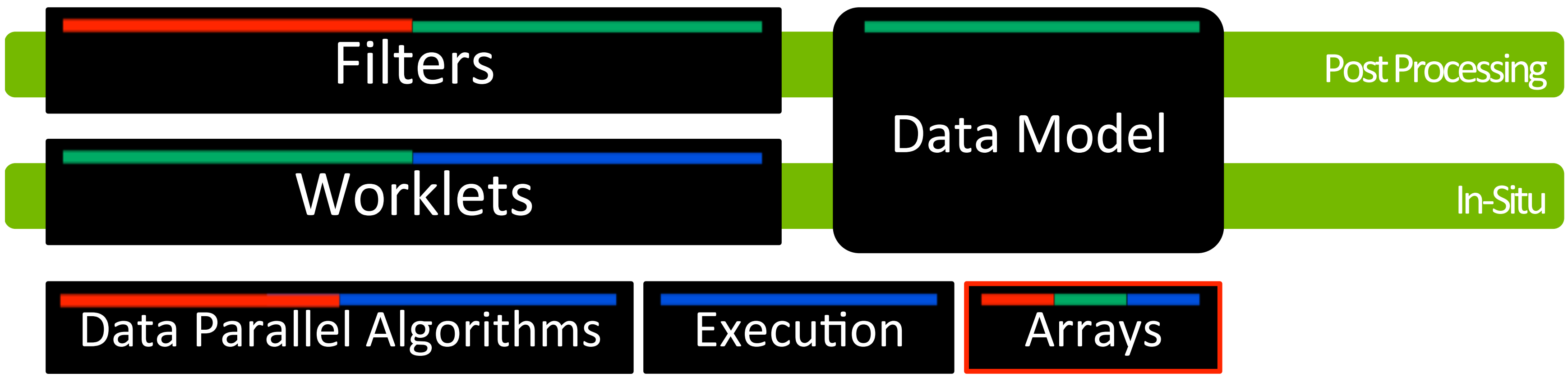
Kewei Lu

Los Alamos NATIONAL LABORATORY EST. 1943
THE OHIO STATE UNIVERSITY

VTK-m

- A toolkit of scientific visualization algorithms for emerging processor architectures
- Support the fine-grained concurrency for data analysis and visualization algorithms by providing abstract models for data and execution
- Can be run across many different processor architectures

VTK-m Architecture

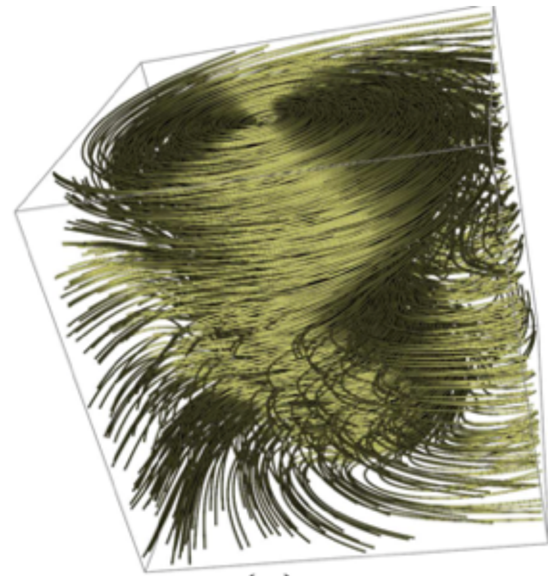


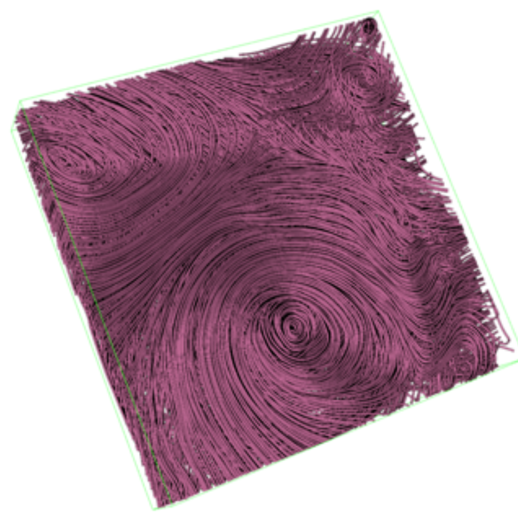
My job this summer

- Based on the current implementation of VTK-m, write different visualization filters:
  - Streamline
  - Stream Surface
- Change the original isosurface implementation using the new data model and worklets in vtk-m
- Measure the performance of those visualization algorithms

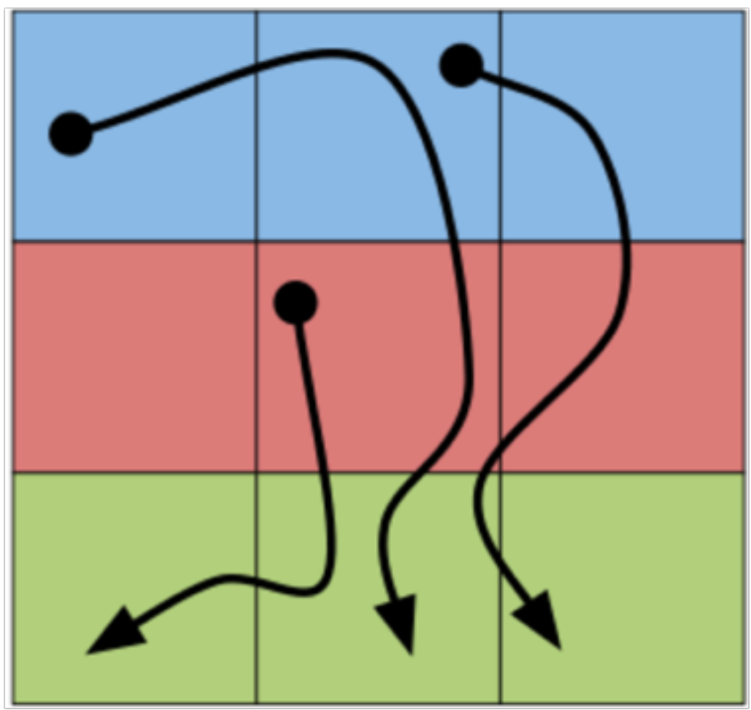
Streamline

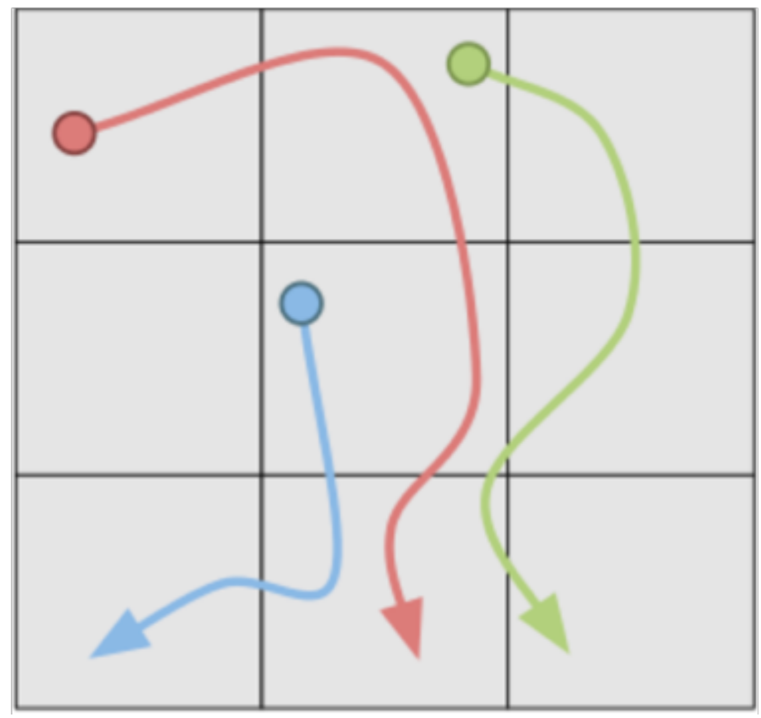
- A curve traced from a particle inside the flow field
- A common method used to visualize and analyze vector fields
- Computation
  - Particle tracing algorithm
  - The fourth-order Runge-Kutta Algorithm





Previous Parallel Streamline Strategies





VTK-m implementation – Streamline(1)

- Adopt parallel-over-seeds approach(map by seeds)
- Algorithm:
  - Read the vector field
  - Randomly generate N seeds
  - Allocate memory for the output streamline buffer(N*maxSteps if only integrate in one direction or N*maxSteps*2 if integrate in both directions)
  - Parallel particle tracing
  - Write the results

VTK-m implementation – Streamline(2)

- Parallel particle tracing

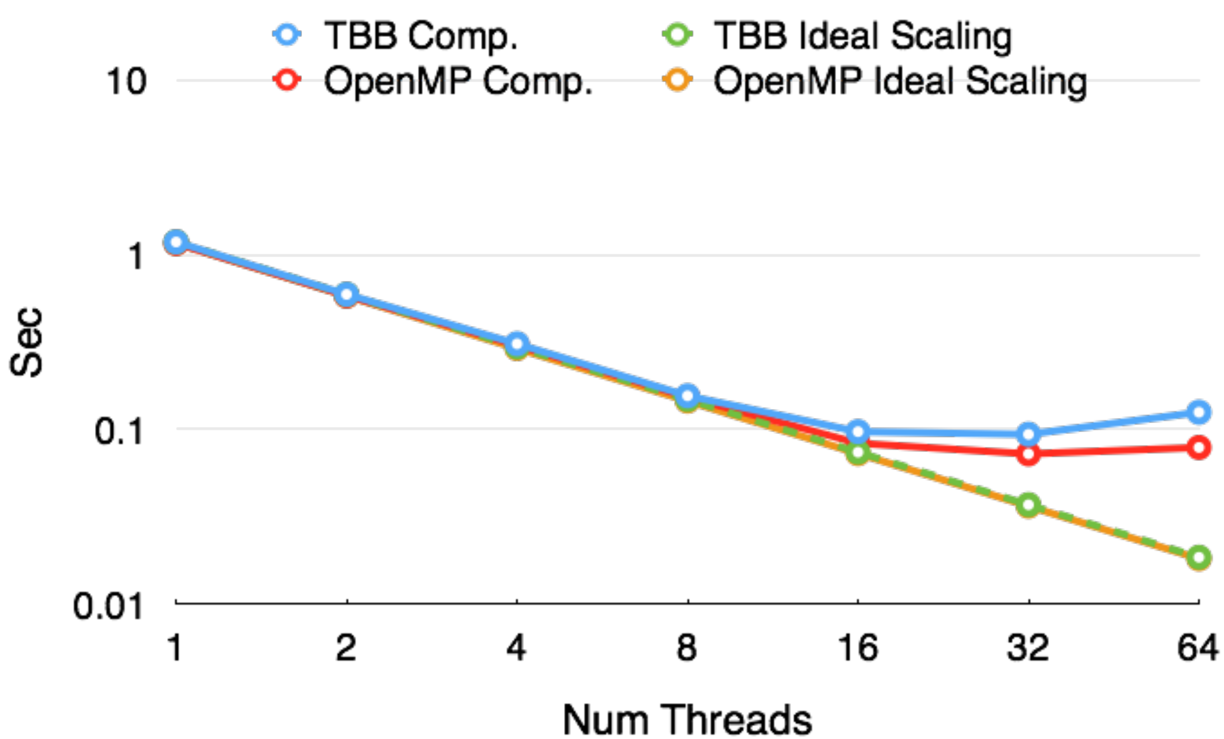
```
        vtkm::cont::ArrayHandle<vtkm::Id> successArray;
        int totalNumParticles=numSeeds*maxSteps*2;
        vtkm::worklet::DispatcherMapField<FieldLineFunctorUniformGrid<FieldType, OutputType> >
fieldLineFunctorDispatcher(FieldLineFunctorUniformGrid<FieldType, OutputType>(t, maxSteps, dim,
 fieldArray.PrepareForInput(DeviceAdapter()), seedsArray.PrepareForInput(DeviceAdapter()),
slLists.PrepareForOutput(totalNumParticles, DeviceAdapter())));
        fieldLineFunctorDispatcher.Invoke(seedIdArray, successArray);
```

VTK-m Performance – Streamline(1)

- Machine: Nvidia partition on Darwin
- Parameters:
  - 100 seeds
  - 2000 steps
- Cuda Timing: 2.85658 sec

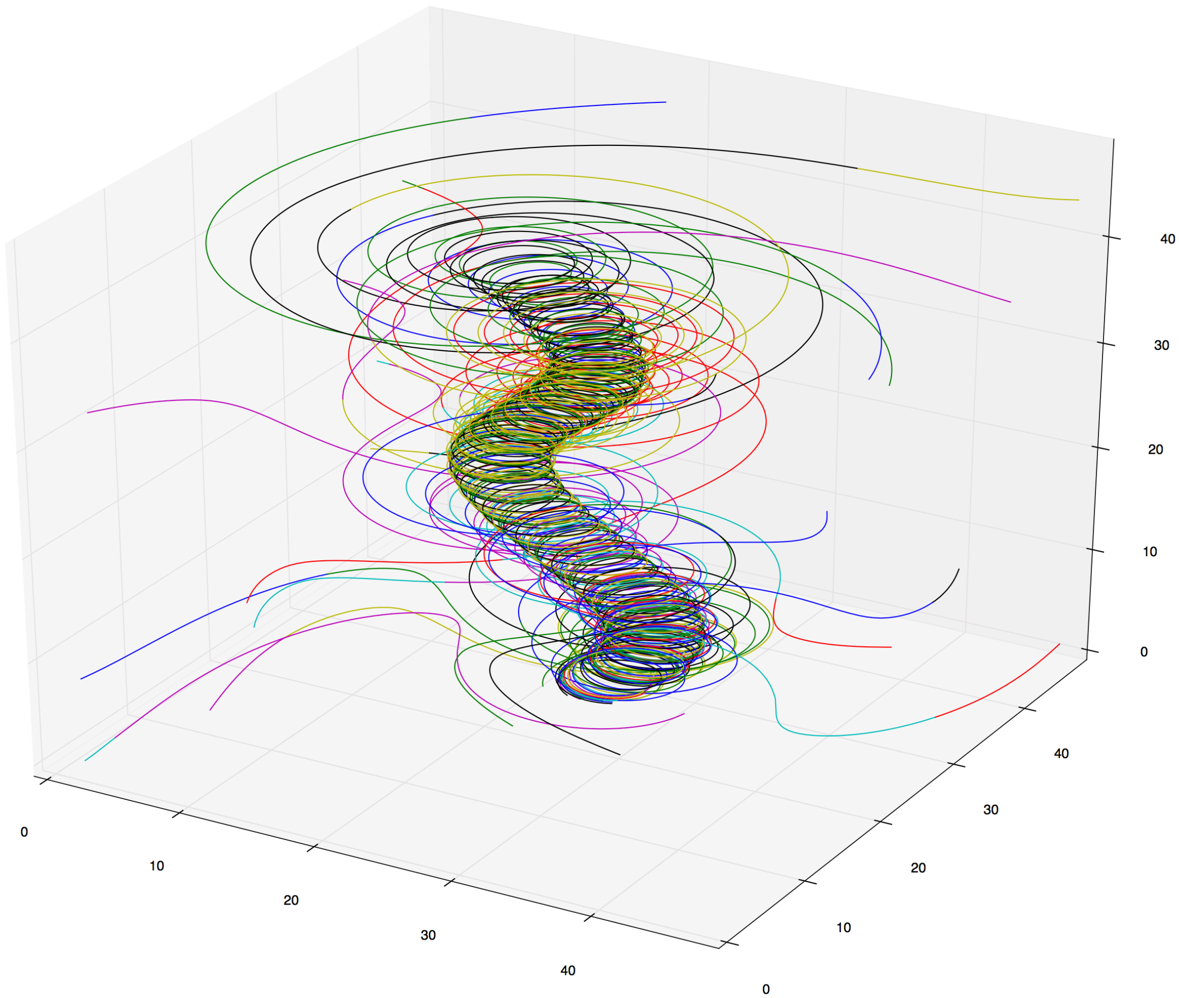
VTK-m Performance – Streamline(2)

- TBB and OpenMP backend



VTK-m Results – Streamline

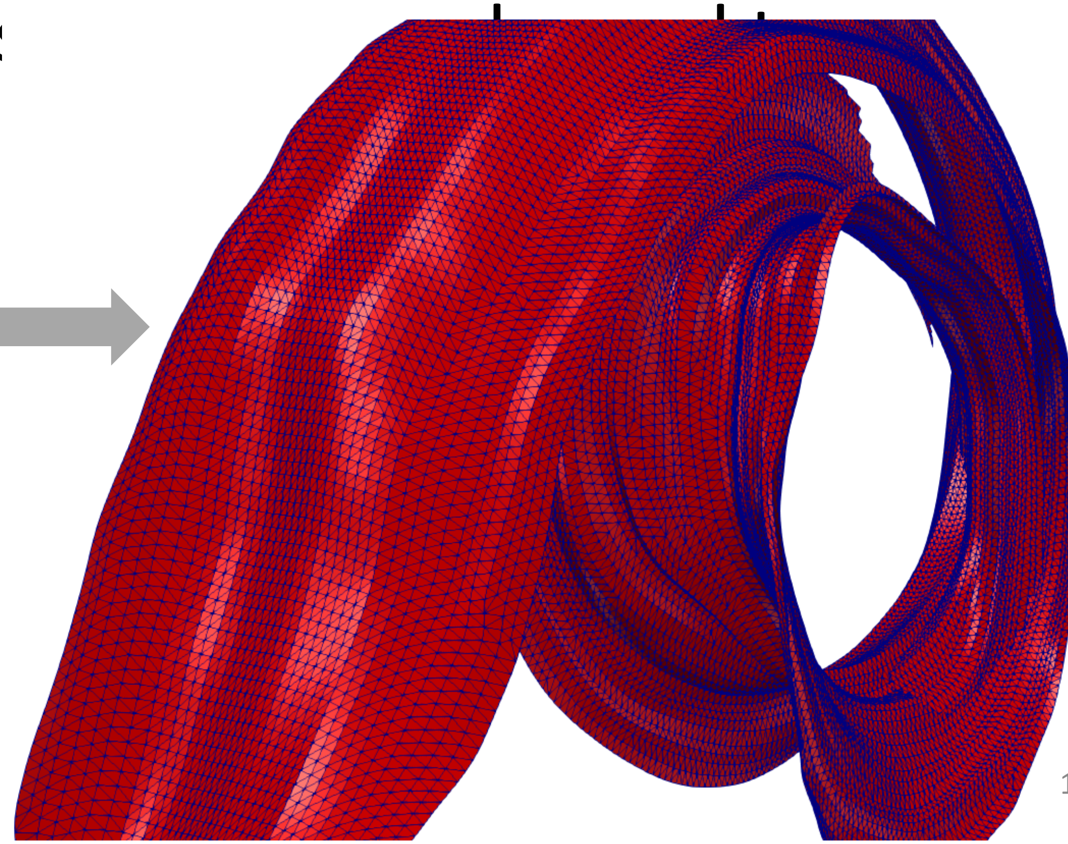
- Results



Stream Surface

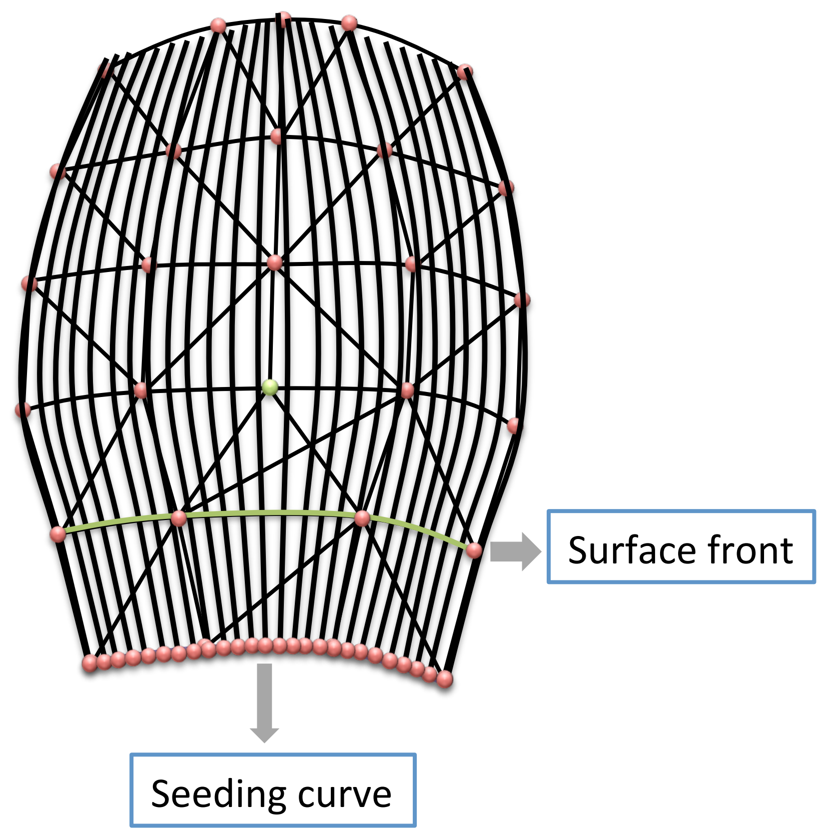
- A stream surface is defined as a surface traced from a seeding curve inside the flow field
- Stream surface
  visualize flow fi

Hurricane Isabel



Stream Surface

- Stream surface
  - The union of an infinite number of streamlines
- Front-advancing algorithm:
  - The seeding curve is discretized
  - Diverge Flow
    - Insert new seeds
  - Converge Flow
    - Delete seeds



VTK-m implementation - Stream Surface (1)

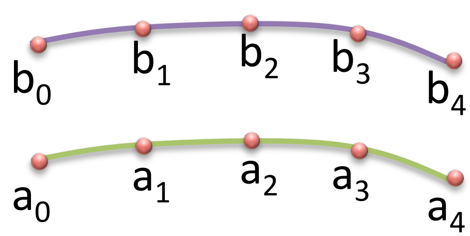
- Goal: A data parallel stream surface algorithm in VTK-m
- Stream surface algorithm:
  - For i from 1 to maximum steps:
    1. Advection
    2. Time line refinement
  - Triangulation

VTK-m implementation - Stream Surface (2)

- Advection
  - Map by seeds

```
vtkm::worklet::DispatcherMapField<RK4FunctorUniformGrid<FieldType, OutputType> >
fieldLineFunctorDispatcher    (RK4FunctorUniformGrid<FieldType, OutputType>(t_, g_dim,
fieldArray.PrepareForInput(DeviceAdapter())));

fieldLineFunctorDispatcher.Invoke(seeding_curve_array, next_time_line_array);
```

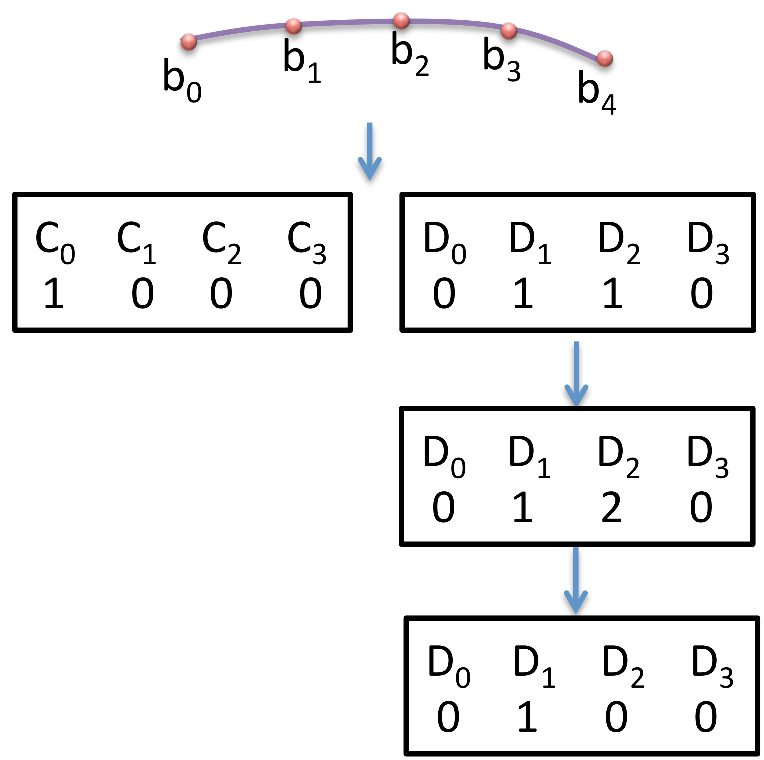


VTK-m implementation - Stream Surface (3)

- Refinement
  - Scatter the new generate particles to current surface front
  - Remove particles from current surface front
  - Algorithm:
    1. Compute the insert and remove decision array
    2. Compute the number of particles on the current surface front after refinement -- allocate space for the output buffer
    3. Compute the decision of every particle on the current surface front, keep the particle or not
    4. Compute the offset of each particle that need to be kept in the output buffer
    5. Scattering
    6. Sort based on particle ID

VTK-m implementation - Stream Surface (4)

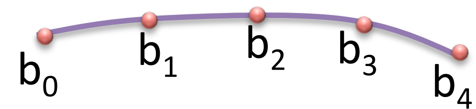
- Step 1
  - Map by every three particles($b_0b_1b_2$, $b_1b_2b_3$, $b_2b_3b_4$) and the last two seeds(b_3b_4)
    - For every three particles($b_{i-1}b_ib_{i+1}$)
      - Whether insert particle between b_{i-1} and b_i and whether remove seed b_i
    - For the last two particles
      - Whether insert particle in between
  - Two Decision Array:
    - 0: none
    - 1: insert or remove
  - TimeLineRefinementFunctor



VTK-m implementation - Stream Surface (5)

2. Compute the number of particles on the current surface front after refinement -- allocate space for the output buffer
   - ScanExclusive on the the two decision array which returns the number of particles to be inserted M and the number of particles to be removed N
   - The length of current surface front after refinement = the length of current surface front + M - N

3. Compute the decision of every particle on the current surface front, keep the particle or not
   - D_0 -> b_0, always 1
   - $D_i \rightarrow b_{i+1}$
     - $1-D_i$



C_0	C_1	C_2	C_3
1	0	0	0

D_0	D_1	D_2	D_3
0	1	0	0

5+1-1=5

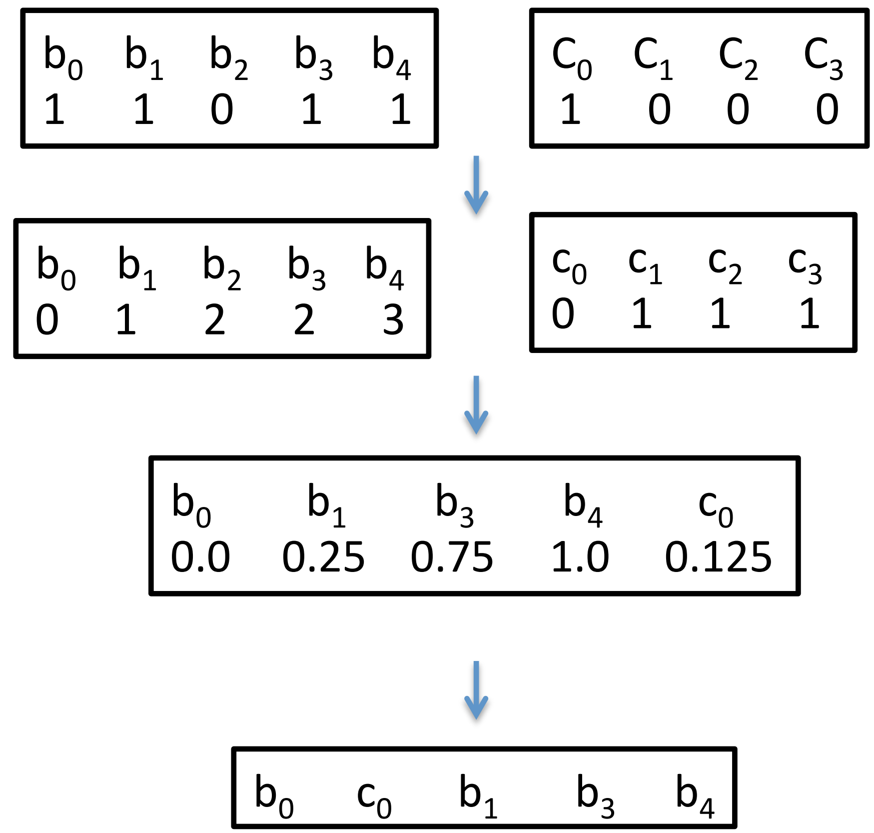
b_0	b_1	b_2	b_3	b_4
1	1	0	1	1

VTK-m implementation - Stream Surface (6)

4. Compute the offset of each particle that need to be kept in the output buffer
   – ScanExclusive on array B and C

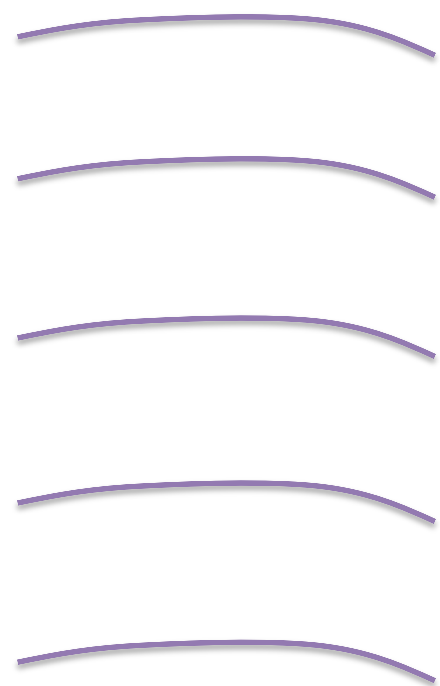
5. Scattering
   – Merge array B and C based on decision and offset
   – MergeToRefinedFunctor

6. Sort based on particle ID
   – vtkm::cont::DeviceAdapterAlgorithm<DeviceAdapter>::SortByKey



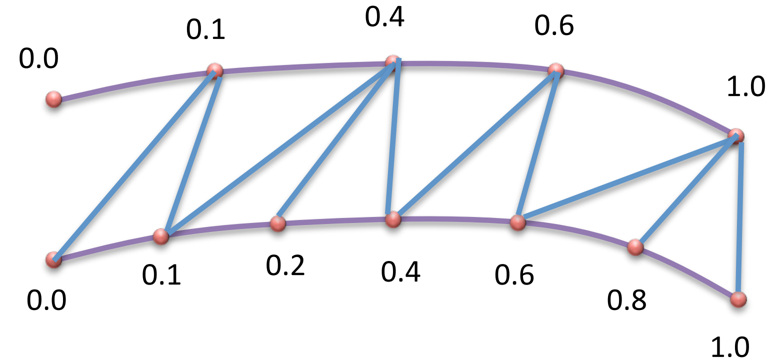
VTK-m implementation - Stream Surface (7)

- Triangulation
  - Input: A number of time lines
  - Output: Triangle mesh(connectivity)
- Mapping
  - Map by every pair of time lines
  - How many triangles each pair will generate?
    - Where to write in the output buffer
    - Suppose one has p_0 particles and the other one has p_1 particles
      - p_0+p_1-2



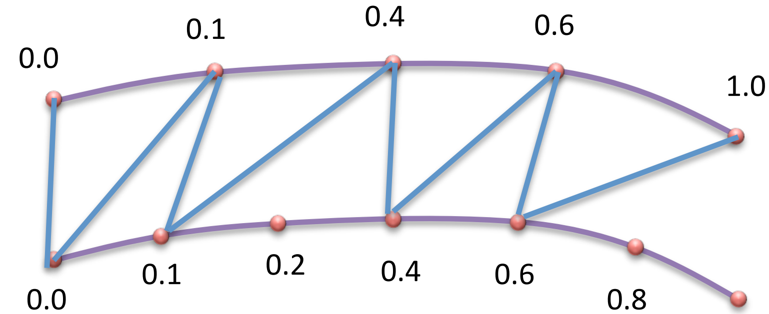
VTK-m implementation - Stream Surface (8)

- Triangulation
  - Bottom line
    - For every two particle, figure out which particle on the top line should be connected
      - Connect to the top particle with particle id bigger or equal to the right particle in the bottom line



VTK-m implementation - Stream Surface (9)

- Triangulation
  - Top line
    - For every two particle, figure out which particle on the bottom line should be connected
      - Connect to the bottom particle with particle id smaller or equal to the left particle in the top line



VTK-m Performance - Stream Surface

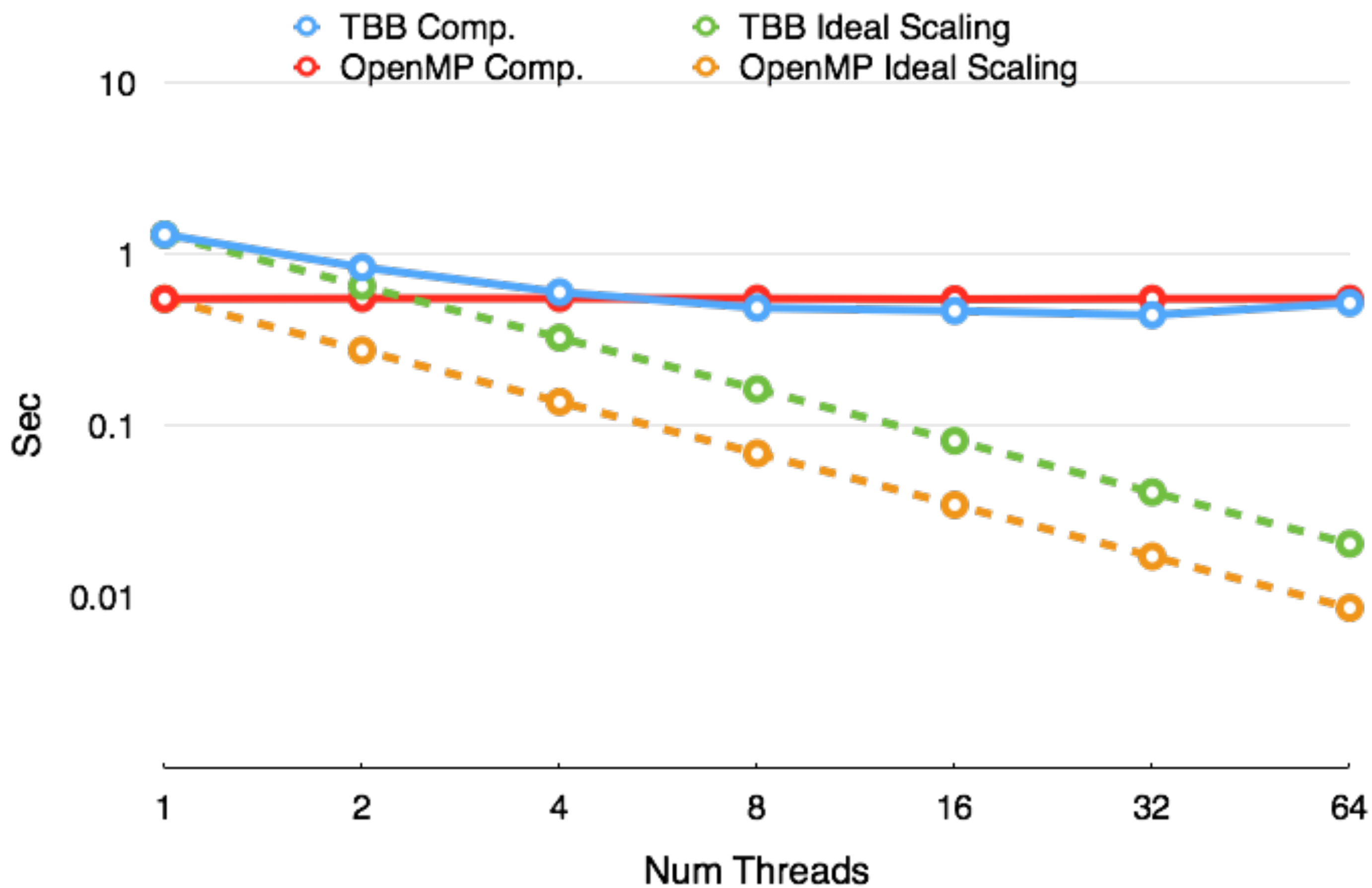
- Machine: Nvidia partition on Darwin

- Parameters:
  - 10 seeds
  - 250 steps

- Cuda Timing: 6.48537 sec

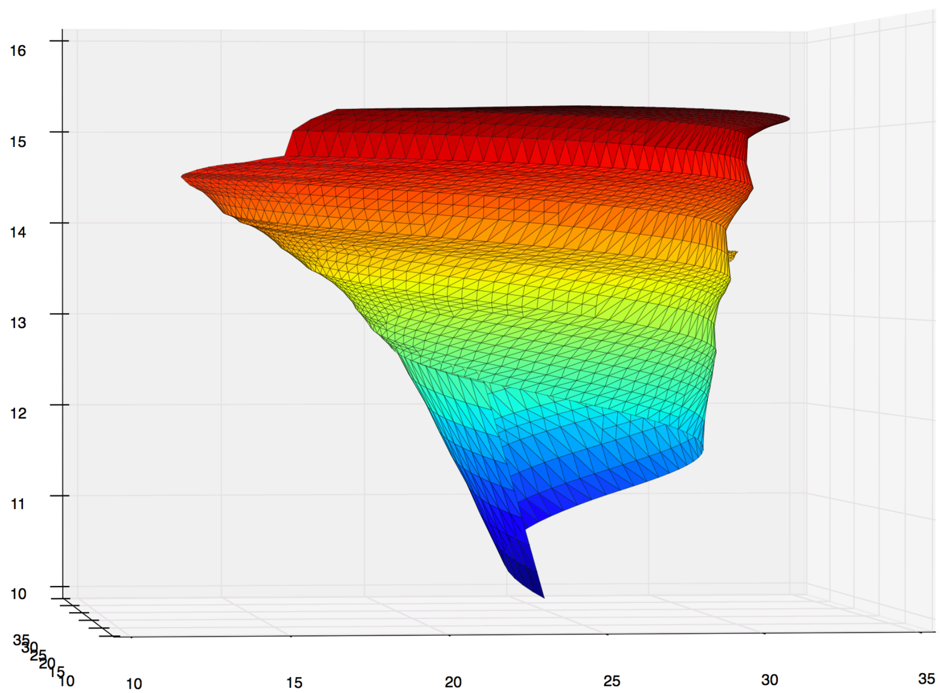
VTK-m Performance - Stream Surface

- TBB and OpenMP backend



VTK-m Results - Stream Surface

- Results



VTK-m implementation – Isosurface(1)

- Original Algorithm: without new data model and topology worklet
  1. Read Data
  2. Classify Cell
     - Determine the case number for each cell
  3. Determine which cell is valid
  4. Compute the write buffer offset for each valid cell
  5. Compute vertices, normal etc.

VTK-m implementation – Isosurface(2)

- New things in VTKm
  - A new DataSet class
  - A new WorkletMapTopology class
- Rewrite the isosurface algorithm using these two new classes

VTK-m implementation – Isosurface(3)

- New algorithm:
  1. Read Data to DataSet class

```
MakeDataSet<FieldType> make_ds(dim);
vtkm::cont::DataSet ds;
if (fileName != 0)
  ds = make_ds.Make3DRegularDataSet0(fileName);
else
  ds =  make_ds.Make3DRegularDataSet0();
```

  2. Classify Cell
     - Determine the case number for each cell
     - Using the new DataSet and WorkletMapTopology class

```
vtkm::cont::ArrayHandleCounting<vtkm::Id> cellCountImplicitArray(0, dim3);
vtkm::worklet::DispatcherMapTopology<ClassifyCell> classifyCellDispatcher(ClassifyCell(vertexTableArray.PrepareForInput(DeviceAdap
ter()), isovalue, verticesPerCellArray.PrepareForOutput(dim3, DeviceAdapter()) ));
vtkm::cont::Field f1("outcellvar", 1, vtkm::cont::Field::ASSOC_CELL_SET, std::string("cells"), vtkm::Float32());
ds.AddField(f1);
classifyCellDispatcher.Invoke(ds.GetField("cellvar").GetData(), ds.GetField("nodevar").GetData(), cs->GetNodeToCellConnectivity(),
ds.GetField("outcellvar").GetData());
```

VTK-m implementation – Isosurface(4)

3. Compute the write buffer offset for every cell

```
    unsigned int numTotalVertices = vtkm::cont::DeviceAdapterAlgorithm<VTKM_DEFAULT_DEVICE_ADAPTER_TAG>::ScanExclusive(verticesPerCell
Array, cellIndicesArray);
```

4. Compute vertices, normal etc.
   - Using the new DataSet and WorkletMapTopology class

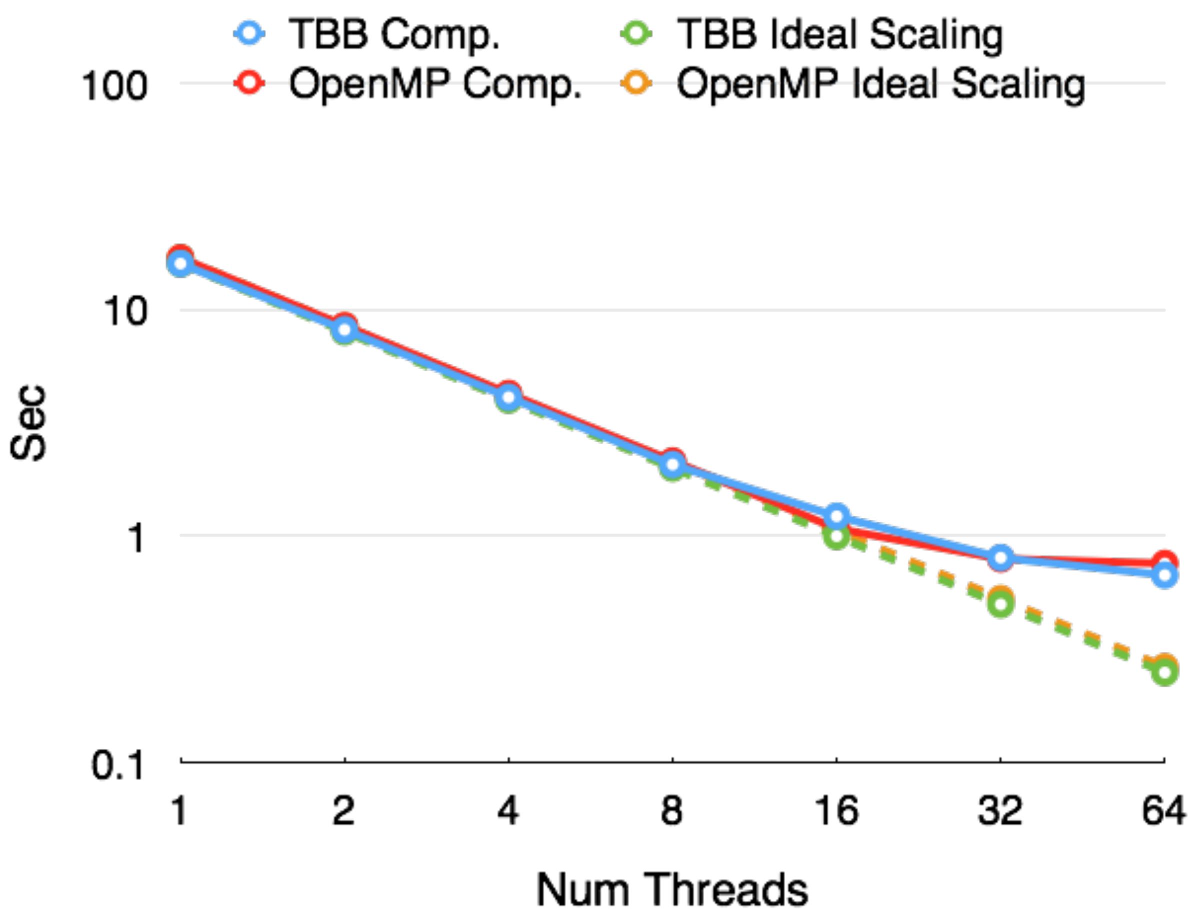
```
    vtkm::cont::ArrayHandle<vtkm::Float32> cellCaseIndex = ds.GetField("outcellvar").GetData().CastToArrayHandle(vtkm::Float32(),VTKM_DEFAULT_STORAGE_TAG ());
    vtkm::worklet::DispatcherMapTopology<IsosurfaceFunctorUniformGrid<FieldType, OutputType> > isosurfaceFunctorDispatcher(IsosurfaceFunctorUniformGrid<FieldType, OutputType>
(isovalue, vdims, mins, maxs,
                                                                    cellIndicesArray.PrepareForInput(DeviceAdapter()),
                                                                    verticesPerCellArray.PrepareForInput(DeviceAdapter()),
                                                                    cellCaseIndex.PrepareForInput(DeviceAdapter()),
                                                                    triangleTableArray.PrepareForInput(DeviceAdapter()),
                                                                    verticesArray.PrepareForOutput(numTotalVertices,DeviceAdapter()),
                                                                    normalsArray.PrepareForOutput(numTotalVertices, DeviceAdapter()) ));
    vtkm::cont::Field f2("outcellsucess", 1, vtkm::cont::Field::ASSOC_CELL_SET, std::string("cells"), vtkm::Float32());
    ds.AddField(f2);
    isosurfaceFunctorDispatcher.Invoke(ds.GetField("cellvar").GetData(), ds.GetField("nodevar").GetData(), cs->GetNodeToCellConnectivity(), ds.GetField("outcellsucess").GetDa
ta());
```

VTK-m Performance - Isosurface

- Machine: Nvidia partition on Darwin
- Parameters:
  - Data size: 200X200X200
  - Isovalue: 0.5
- Cuda Timing: 0.029479 sec

VTK-m Performance - Isosurface

- TBB and OpenMP backend



Summary

- Streamline and Stream Surface filters for vtkm
- Rewrite the isosurface filter using the new data model and worklet
- Performance measurement

Acknowledgement

- Mentor: Christopher Sewell and Li-Ta Lo
- James Ahrens, Curtis Canada, Erika Maestas